Position Paper: *ARO Workshop Security of Embedded Systems and Networks*

Sean Smith
Dartmouth College
`http://www.cs.dartmouth.edu/~sws/`

February 4, 2007

I'll start with two caveats. The first is a matter of scale. Trying to write this position paper leads me to a conundrum. The workshop's grand overture poses problems and issues that excite me, and spark research ideas. "Hey, maybe we could address problem Z by combining techniques X and Y!" However, that is the stuff of a specific proposal (and might become one, if my colleagues cooperate); the workshop invitation then jars me out of this low-level vision to a high-level one. "What are the three most superlative...."

- **Caveat 1:** It's hard to distinguish the "big picture" from the smaller details.

The second is a matter of tone. Before I came back to academia, I spent time working for the government (advising folks worried about security in real-world deployments) and then industry (building security-motivated embedded systems in the real worlds). But academics often value "intellectual depth" over the real world.

- **Caveat 1:** It's hard to distinguish the "fundamental science" from what's merely an important problem or promising tool for a real-world problem.

Fundamental limitations of today's security mechanisms

Not Thinking about Enough Levels. My generation came of age at a time when the only way for a young teen to get a computer was to build one—but in those early days of 8-bit microprocessors, one actually *could.* History forced a computing education that started at almost the transistor level and worked its way up. This approach had downsides, such as a predilection for C and a parsimony for saving bytes. But it had an upside as well: one starts to see that there are many layers to what we consider as "computation." However, perhaps as a result of the increasing complexity of computing systems, we see too many security mechanisms (for embedded systems—or any other system) focus on only one level.

- Use of typesafe languages may eliminate certain classes of vulnerabilities—but may come at a performance and usability cost (ever try to code in Clay?) and have been successfully compromised via light-bulb-induced memory errors. (Looking only at the language level also neglects how much of the underlying libraries, OS, and firmware may be written in very primitive languages, such as assembler.)
- Hardware techniques such as trusted platform modules and the TCG architecture bind secrets and attestations to a system configuration expressed by a series of hashes. However, mapping from this expression to the properties that relying parties really care about is not obvious (and perhaps not even tractable). What's worse, until the emergence of resettable PCRs, this "configuration" ended up including the whole operating system. Another example of this limitation is the contortions and reworking required to mesh the TPM-based architecture with the full power of virtualization.
- Secure architectures don't guarantee that their applications won't have flawed APIs. (As a member of the 4758 team, I know this one from personal experience.)
- Hardware tokens that protect private keys fail to take into account the integrity and authentication of the software systems that request their use.
- Thinking about computation solely in the logical terms of a box with binary inputs and outputs neglects the fact that the box functions as a physical device in the physical world. This oversight has led to over a decade of side-channel attacks in the open world—and even perhaps to last week's fun of using audio content on Web sites to launch privileged commands on Vista.
- Thinking about a hardware device as some set of modules specified in VHDL or Verilog neglects to take into account that whether the netlists and layout that comes out at the end correctly embody the specification that went in.

- The recent calls for a "Cyber Manhattan project" seem to implicitly assume that the computer security crisis can be solved with technology alone. However, success requires navigating human, policy, and economic angles as well. The original Manhattan project only had to build a few bombs—it didn't have to change the way all of humanity used refrigerators.

One might even go out on a limb and say that we need a new kind of composition theorem: not between "peer" modules, but rather across layers and organizational boundaries.

Cryptography's Questionable Future. Sending a computing system out into the cold, cruel world is scary—particularly for embedded devices, which often must fend for themselves. Security architectures to control code updates and to authenticate devices are based on cryptographic protocols. Best practices tell how to soundly engineer protocols from the primitives and keylengths that cryptographers deem secure. However, the security of these primitives are themselves based on assumptions. History has shown these assumptions don't always remain true as long as anticipated. Here one cite ambitious predictions about RSA and DES; the slow decline of MD5 over ten years; or the lesser-noticed rapid decline of the ISO9796 padding function. The elephant on the table now is SHA1; the TCG meetings just last week were full of heated discussion on how to add "hash agility" to the TPM specification—and the pushback from businesses who don't want to commit product hardware to a specification that is proving malleable.

Keeping Secrets. Sending a self-protecting embedded system out into the cold, cruel world also often requires that the system itself be able to keep and use cryptographic secrets. Here again, history shows that keeping secrets is difficult. In the open world, one need only look at Ross Anderson's lab for a good sampling of how researchers have made repeated hash of low-cost "tamper-protected devices" and exploited amusing side-channels. A few promising techniques have come along—our 4758 device has yet to have its core protections broken (to our knowledge), but is rather expensive and brittle; the SPUF work from MIT looks interesting. However, a method by which inexpensive, environmentally durable devices can keep secrets with high assurance—and a method to have assurance that they can—eludes us (at least in the public world).

Promising innovations and abstractions for future systems

TCG. I could go on for a long time on all the problems with the TCG organization and architecture. The standard quips—such as "designed by committee, and it shows"—suffice. However, to paraphrase someone, the important thing here is not that the bear dances well, but that it dances at all. Fifteen years ago, we could play with secure coprocessor prototypes and say "pretend that all commodity devices had something like this built in." Now they do, sort of. Getting an idea to permeate the broader infrastructure is the hard part, but industry has now done that, and looks poised to continue being amenable to such thinking.

Multicore. Perhaps as a consequence of trying to adhere to Moore's Law, hardware vendors are now giving us more cores than we know what to do with—and are desperately searching for applications, usage models, and a business case. This position gives us an opportunity to rethink what a CPU "does"—and to see these changes happen in the real world.

Calls for a principled revolution. Voices over recent years (e.g., at the CRA Grand Challenges in Infosec workshop a few years back, or at the NSF Safe Computing Workshop last fall, or at the Cyber Trust PI meeting last week) have repeated the observation that things aren't working, that the current approaches to securing computing systems aren't working, and that (to paraphrase Rich DeMillo) we're not "likely to program our way out of this mess." The right questions are being asked.

Possible milestones

This part is harder for me. I might joke that it's because I've already pushed the limits of my space budget; however, the reality is that it's hard for me to describe the end result of a research program that hasn't happened yet.

ARO Planning Workshop Security of Embedded Systems and Networks

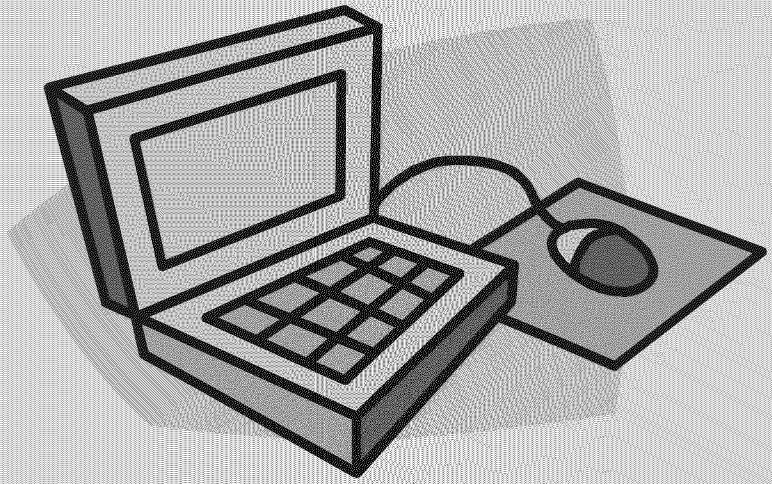
Hardware-Based Security: Trouble and Hope

Sean W. Smith
Department of Computer Science
Dartmouth College
Hanover, NH USA

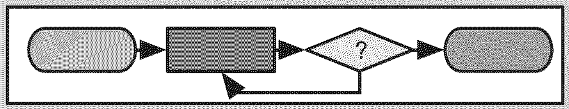
http://www.cs.dartmouth.edu/~sws/

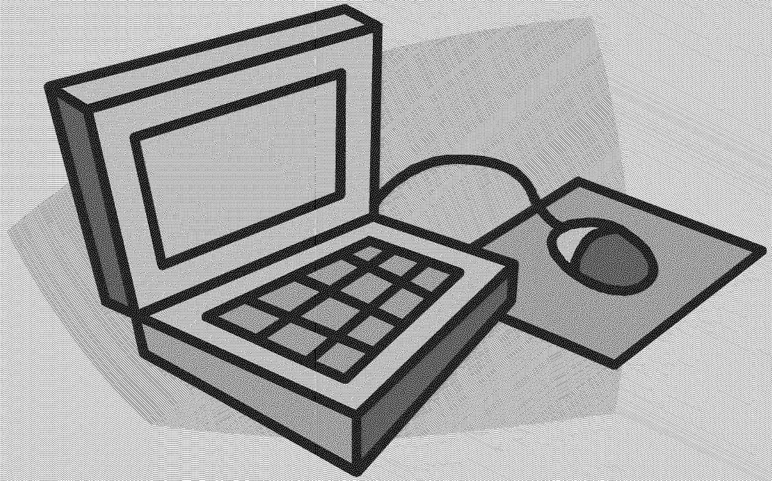
February 22, 2007

Trouble Area #1:
<u>Not Thinking about Enough Levels</u>

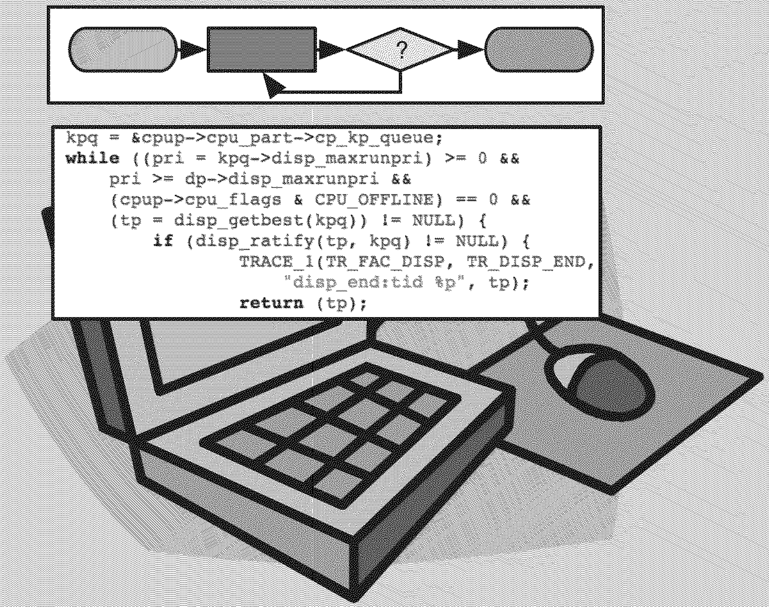


Trouble Area #1:
Not Thinking about Enough Levels





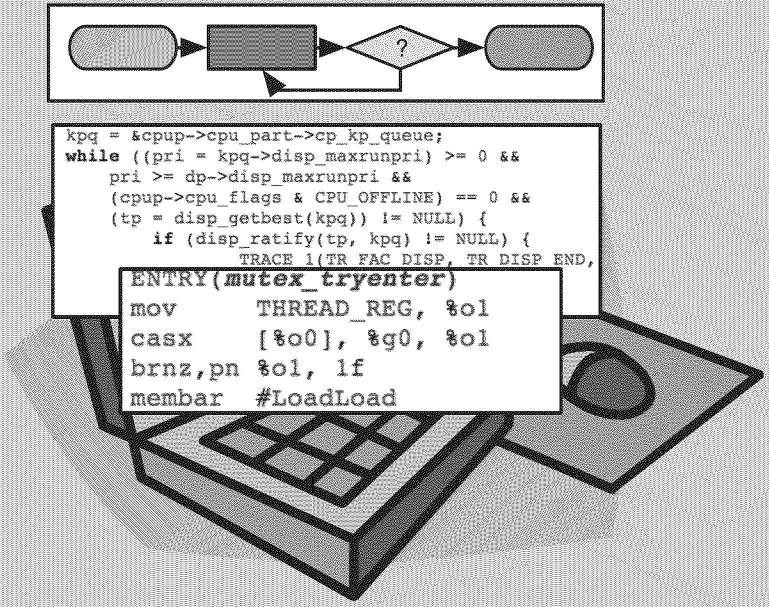
Trouble Area #1:
Not Thinking about Enough Levels



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
                    "disp_end:tid %p", tp);
                return (tp);
```

Trouble Area #1:

Not Thinking about Enough Levels

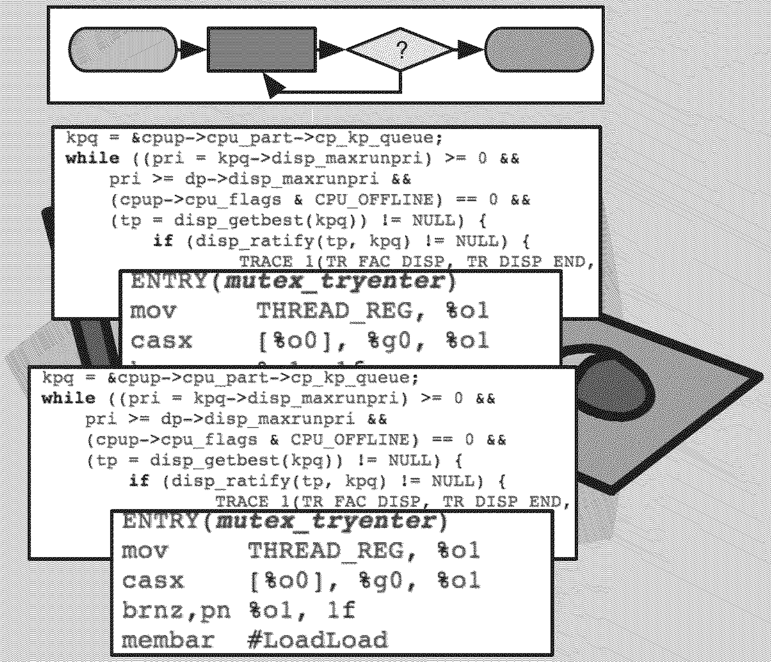


```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels



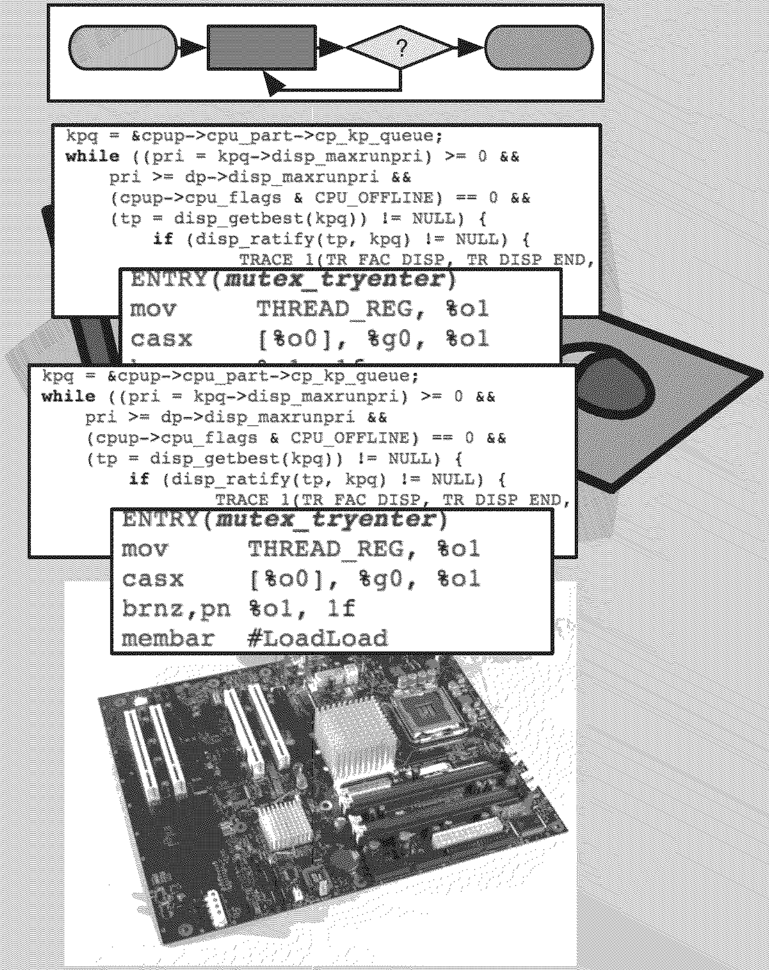
```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
```

```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

Trouble Area #1:
Not Thinking about Enough Levels



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

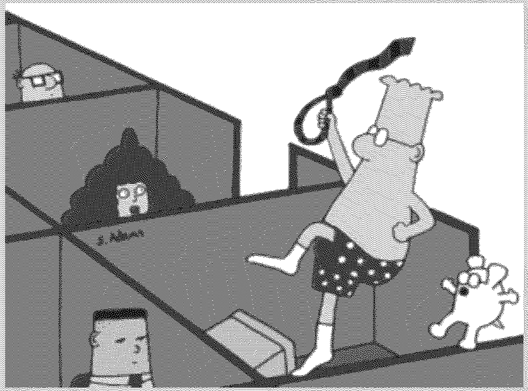
```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
```

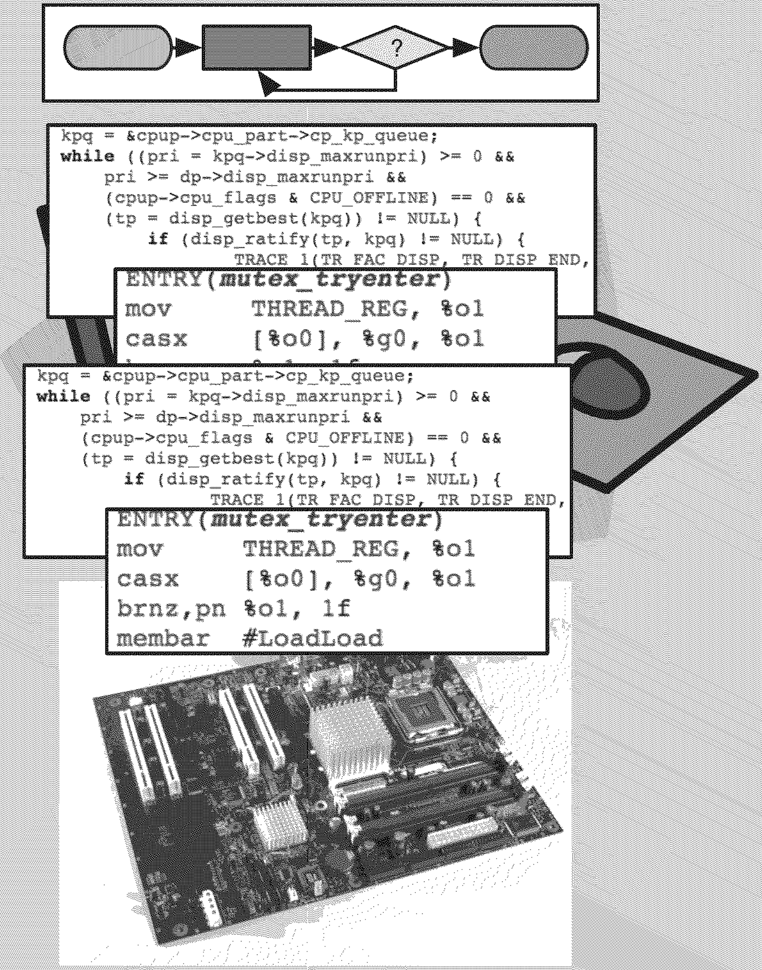
```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
brnz,pn  %o1, 1f
membar   #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels





```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

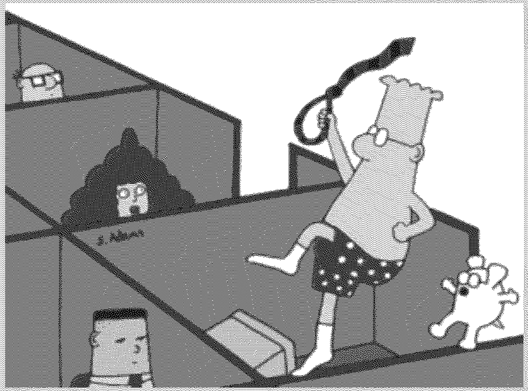
```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
```

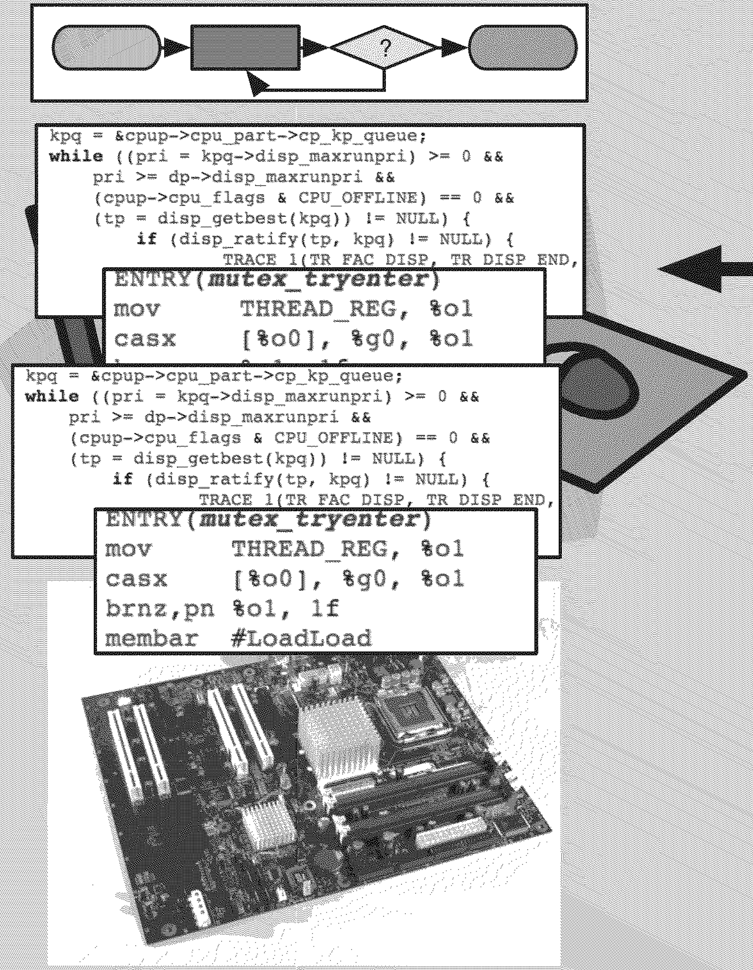
```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
brnz,pn %o1, 1f
membar   #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels





```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

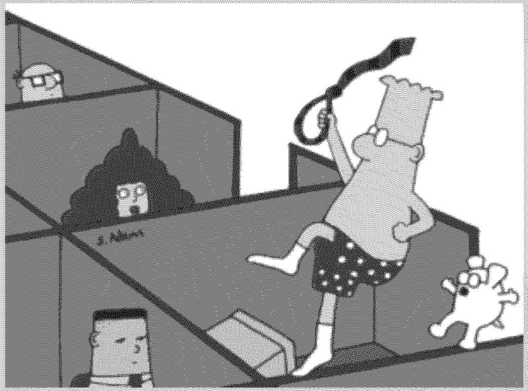
```
ENTRY(mutex_tryenter)
mov     THREAD_REG, %o1
casx    [%o0], %g0, %o1
```

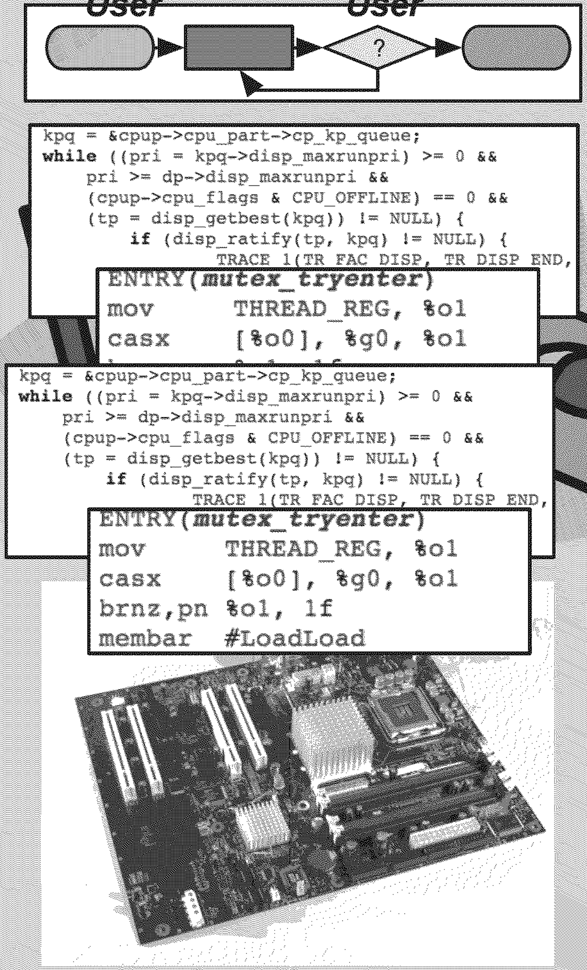
```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov     THREAD_REG, %o1
casx    [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

Trouble Area #1:
Not Thinking about Enough Levels

User User User User User User





```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
```

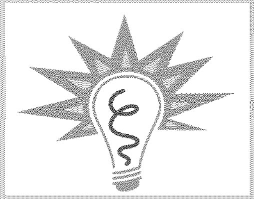
```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
brnz,pn %o1, 1f
membar   #LoadLoad
```

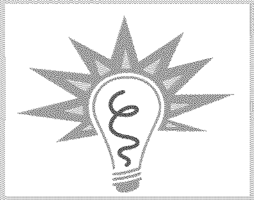
Vox Clamantis in Deserto

Examples

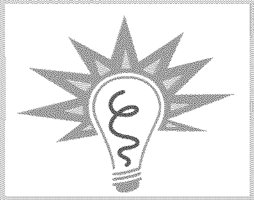
- Java type-safety vs. light bulbs



Examples

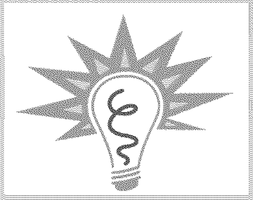
- Java type-safety vs. light bulbs 
- Type-safe C variant for kernel coding vs. kernel coders

Examples

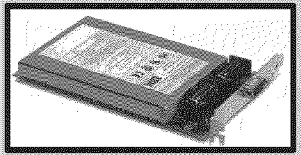
- Java type-safety vs. light bulbs 
- Type-safe C variant for kernel coding vs. kernel coders
- hardware-based attestation vs. the computational entity
- hardware-based attestation vs. the operating system

Examples

- Java type-safety vs. light bulbs



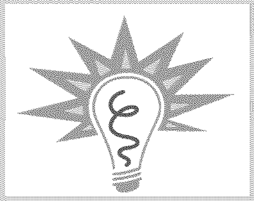
- Type-safe C variant for kernel coding vs. kernel coders
- hardware-based attestation vs. the computational entity
- hardware-based attestation vs. the operating system



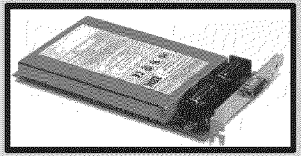
- IBM 4758 platform vs. API flaws in the CCA app

Examples

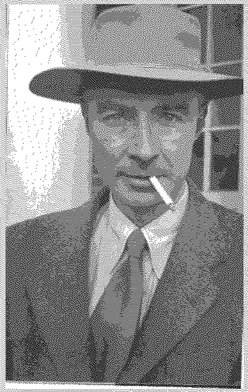
- Java type-safety vs. light bulbs



- Type-safe C variant for kernel coding vs. kernel coders
- hardware-based attestation vs. the computational entity
- hardware-based attestation vs. the operating system



- IBM 4758 platform vs. API flaws in the CCA app
- Cyber-Manhattan project vs. economic roll-out



More Trouble Areas

2. Cryptography's questionable future

More Trouble Areas

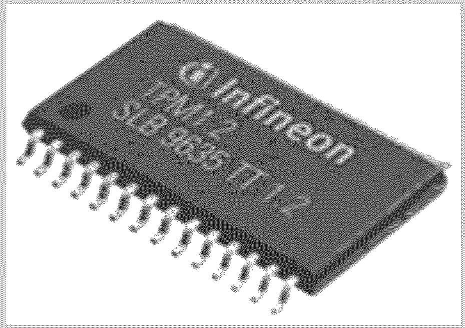
2. Cryptography's questionable future

More Trouble Areas

2. Cryptography's questionable future

3. Keeping and using secrets

- *effectively*
- *affordably*

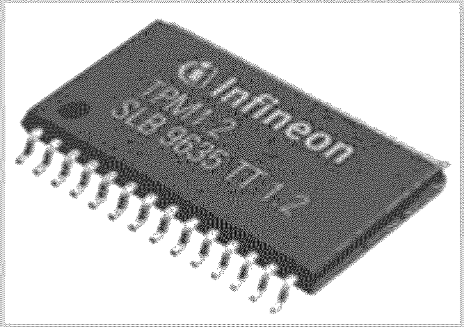


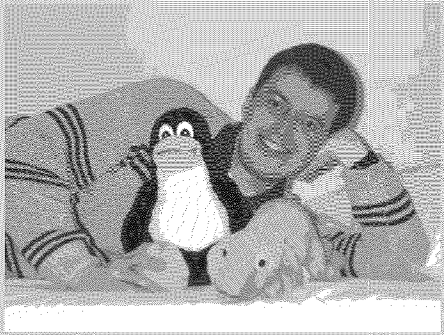
More Trouble Areas

2. Cryptography's questionable future

3. Keeping and using secrets

- *effectively*
- *affordably*





Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED COMPUTING GROUP™

Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED COMPUTING GROUP™

Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED COMPUTING GROUP™

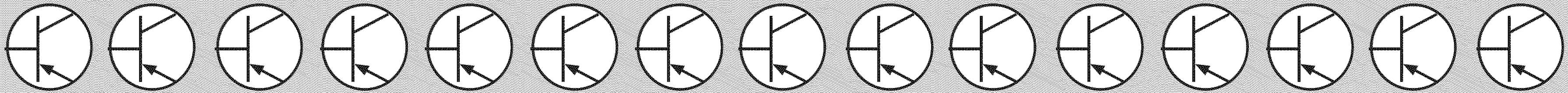
2. The consequences of ***keeping up*** with Moore's Law

Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED COMPUTING GROUP™

2. The consequences of ***keeping up*** with Moore's Law



Vox Clamantis in Deserto

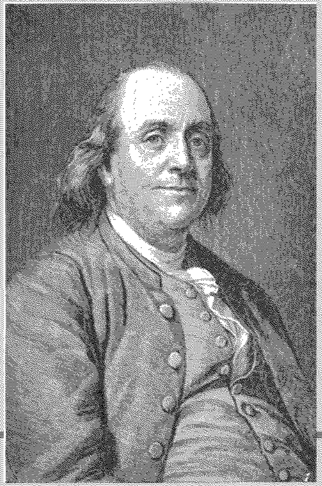
Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

2. The consequences of ***keeping up*** with Moore's Law

3. Repeated calls for ***principled revolution***

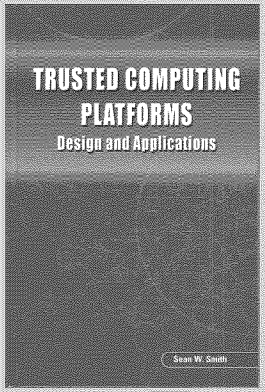
- CRA, I3P, "Cyber-Manhattan Project,"....
- ***This workshop***



Thanks

Sponsors:

- NSF CAREER, DoJ/DHS, Mellon, Internet2/AT&T
- Sun, Intel, Cisco (and IBM Research)



For more information:

- http://www.cs.dartmouth.edu/~sws/
- *Trusted Computing Platforms: Design and Applications*. Springer, 2005.